Capstone Project

Fan Yuan
Jan 9th, 2021

I. Definition

Project Overview

With the mature application of machine learning technology, more and more company are taking the advantage of big data to get more personalized customer experience. As one of the biggest coffee giants in the world, Starbucks operates more than 31,000 stores worldwide, in 2017, Starbucks launched its most advanced AI-driven initiative "Deep Brew". The brand's custom-made recommendation platform was built to reach customers across multiple channels, including the Starbucks ordering app. With the platform, it delivers highly customized offers to the almost 19 members of its My Starbucks Rewards loyalty program. It helps Starbucks to deploy individualized offers across channels by automating offer assembly and management, reward fulfillment, and KPI measurement and tracking, at enterprise scale.

The industry-leading Starbucks Rewards Program has continued to flourish since its introduction in 2007. According to their website, "Membership has grown more than 25% over the past two years alone, climbing to 16 million active members as of December 2018, a 14% increase over the prior year. Starbucks Rewards transactions accounted for 40% of tender in U.S. company-operated stores in the same time frame." Obviously, it becomes more and more important to leverage the AI technology to enhance the customer experience.

Problem Statement

Since it's important to provide customized experience from ordering to offer on loyalty app, the problem becomes to how to correctly and precisely locate the target customer group. That is, the task is to decide what the best-personalized offer is to send to users so that the conversion rate can be maximized. Also, it's critical to know which groups of people are most responsive to each type of offer, and how best to present each type of offer.

To solve the problem stated above, this project is to build a machine learning model to decide which is the best type of offer to send to each customer. The dataset will be separated into three types of offer and fit the data into three supervised classification models. Using this way, the model will predict whether the offer will be responded by customer or not when sent to them. Also, by investigating the feature importance of the model, it can help answer the question that what factors mainly affect people to make the decision and finally complete the transaction. Therefore, with the solution above, it'll be easier and more accurate to identify which groups of people are most responsive to each type of offer, and how best to present each type of offer.

The project will be divided into several steps:

- Prepare and clean data -- combine transaction, demographic and offer data. Understand the connection between columns and dataset. Try to get the useful information from data as much as possible.
- Data exploration -- In order to analyze the problem better in next sections, first need to explore the datasets which includes checking the missing value, visualizing the data distribution, etc.
- Data preprocessing -- In order to find out what mainly affect the finish of the transaction by sending the offer, in the data processing process, also need to process the data to merge the events of each specific offer sent so as to find out which offer were received, viewed and finally completed with a transaction.
- Feature engineering -- After basic processing, the next step will look if there are any columns that can be used to create new features. For example, generating a new column for length of customer's membership, the count of offer received for each user, calculate the time lap between offers, etc.
- Building model – after pre-processing and feature engineering, next step is to build the model using response flag generated in previous steps to predict whether the customer will respond to the offer or not.
- Model tuning – Compare the model using metrics selected above and tune the parameters of initial model using GridSearch method to get higher performance.
- Conclusion and further improvement – compare the final selected model to benchmark to see if the solution provide a better personalized offer. Also, review the built process and see if there's any opportunities to enhance the model in the future.

Metrics

Since the project is building classification model, here choose both accuracy and F1 score as the model evaluation metric. The reason of choosing both metrics is sometimes when the dataset is imbalanced, the accuracy only couldn't objectively show how the model is performing on the dataset, while F1 score provides a better sense of model performance compared to purely accuracy as takes both false positives and false negatives in the calculation. With an imbalanced class distribution, F1 may be more useful than accuracy.

Also, since the F1 score is based on the harmonic mean of precision and recall and focuses on positive cases. For the Starbucks app here, it would be fine as we would prioritize more on whether offers are effective, and less focus on why offers are ineffective.

II. Analysis

Data Exploration

The dataset to be used in this project contains three files. The profile dataset contains Rewards program user's information which gives data like gender, age, income and time when the customer became a member. The portfolio dataset contains the list of all categories of offers sent to the customer during 30-day test period. There are three types of offers that can be sent: buy-one-get-one (BOGO), discount, and informational. And the last dataset is transcript which gives event showing different actions (e.g., offer received, offer viewed) and amount of the transaction spent on the offer.

In order to analyze the problem better in next sections, first need to explore the datasets which includes checking the missing value, visualizing the data distribution, etc. In that way, we can have a better understanding on how the dataset looks like and how to select the important features to support the model implementation.

Within portfolio dataset, there are 10 unique offer types with different channels, difficulty (minimum required spend to complete an offer), duration (time for offer to be open, in days) and reward. Counting the number of different categories in offer_type column, there are 4 BOGO offers, 4 discount offers and 2 informational offers. After quick review, there's no missing value in the portfolio dataset.

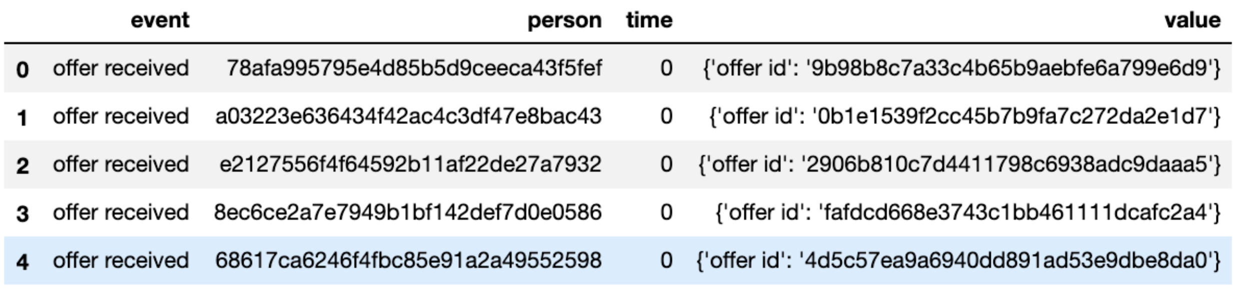
	channels	difficulty	duration	id	offer_type	reward
0	[email, mobile, social]	10	7	ae264e3637204a6fb9bb56bc8210ddfd	bogo	10
1	[web, email, mobile, social]	10	5	4d5c57ea9a6940dd891ad53e9dbe8da0	bogo	10
2	[web, email, mobile]	0	4	3f207df678b143eea3cee63160fa8bed	informational	0
3	[web, email, mobile]	5	7	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	5
4	[web, email]	20	10	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	5
5	[web, email, mobile, social]	7	7	2298d6c36e964ae4a3e7e9706d1fb8c2	discount	3
6	[web, email, mobile, social]	10	10	fafdcd668e3743c1bb461111dcafc2a4	discount	2
7	[email, mobile, social]	0	3	5a8bc65990b245e5a138643cd4eb9837	informational	0
8	[web, email, mobile, social]	5	5	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5
9	[web, email, mobile]	10	7	2906b810c7d4411798c6938adc9daaa5	discount	2

Next, look at profile dataset. There are 17000 unique customer information within dataset, and there are a number of missing values in age column which is encoded as 118. By filtering out the entries which have 118 in age columns, there are 2175 missing ages, which also have missing values in both gender and income columns. Since it doesn't account big proportion of the whole dataset (12% in terms of all entries in the dataset), these entries can be removed in later steps. And quickly look at the statistics summary of age and income features both of which have normal distribution.

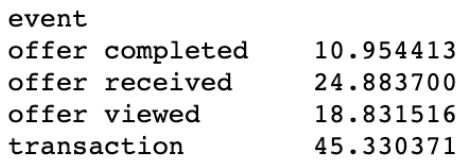
	age	became_member_on	gender	id	income
0	118	20170212	None	68be06ca386d4c31939f3a4f0e3dd783	NaN
2	118	20180712	None	38fe809add3b4fcf9315a9694bb96ff5	NaN
4	118	20170804	None	a03223e636434f42ac4c3df47e8bac43	NaN
6	118	20170925	None	8ec6ce2a7e7949b1bf142def7d0e0586	NaN
7	118	20171002	None	68617ca6246f4fbc85e91a2a49552598	NaN

	age	became_member_on	income
count	17000.000000	1.700000e+04	14825.000000
mean	62.531412	2.016703e+07	65404.991568
std	26.738580	1.167750e+04	21598.299410
min	18.000000	2.013073e+07	30000.000000
25%	45.000000	2.016053e+07	49000.000000
50%	58.000000	2.017080e+07	64000.000000
75%	73.000000	2.017123e+07	80000.000000
max	118.000000	2.018073e+07	120000.000000

The last dataset is transcript which contains records for transactions, offers received, offers viewed, and offers completed. The dataset has 306534 records with 4 columns. Except transaction entries which accounts a big amount of data, the other three types of event have similar proportion within the dataset.



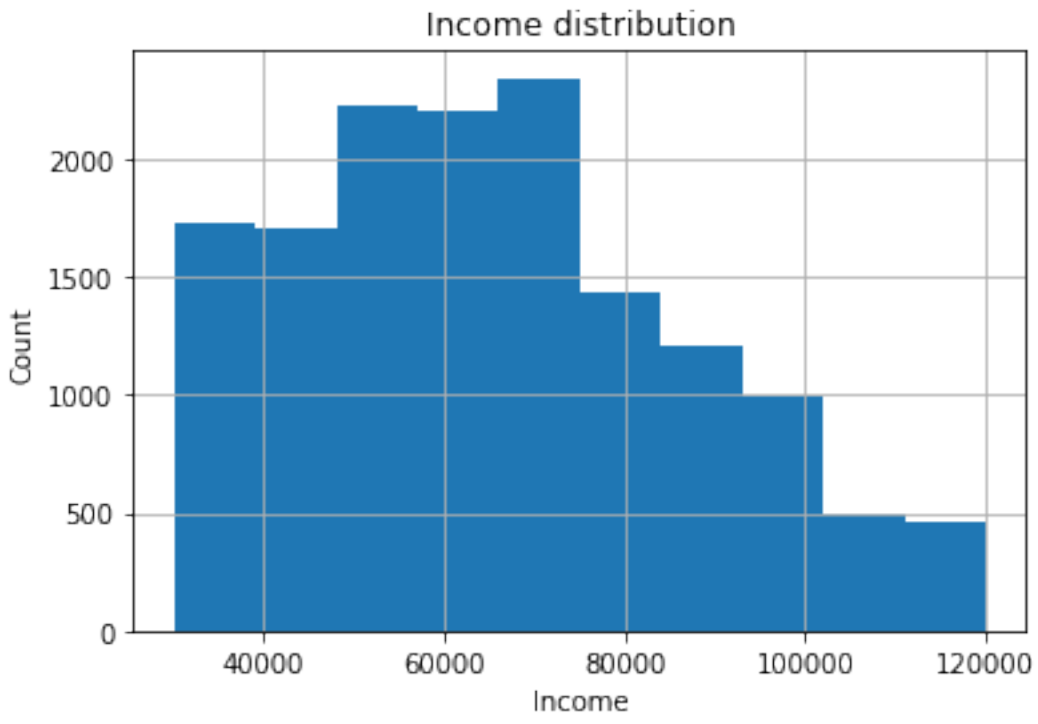
	event	person	time	value
0	offer received	78afa995795e4d85b5d9ceeca43f5fef	0	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}
1	offer received	a03223e636434f42ac4c3df47e8bac43	0	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
2	offer received	e2127556f4f64592b11af22de27a7932	0	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}
3	offer received	8ec6ce2a7e7949b1bf142def7d0e0586	0	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}
4	offer received	68617ca6246f4fbc85e91a2a49552598	0	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}



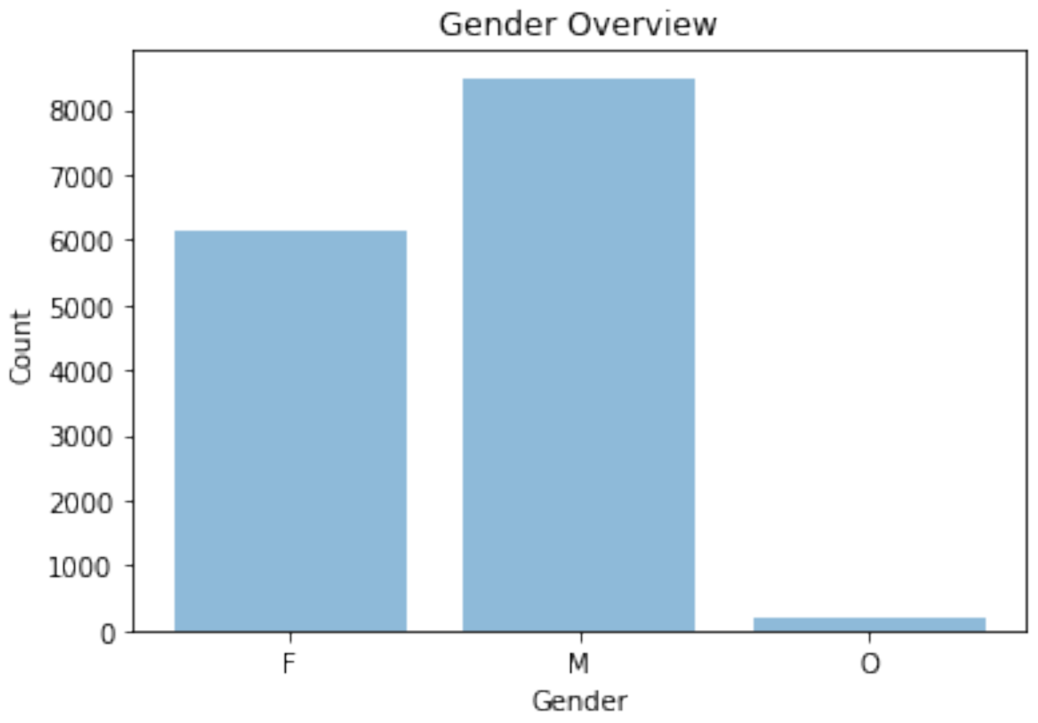
```
event
offer completed    10.954413
offer received     24.883700
offer viewed       18.831516
transaction        45.330371
```

Exploratory Visualization

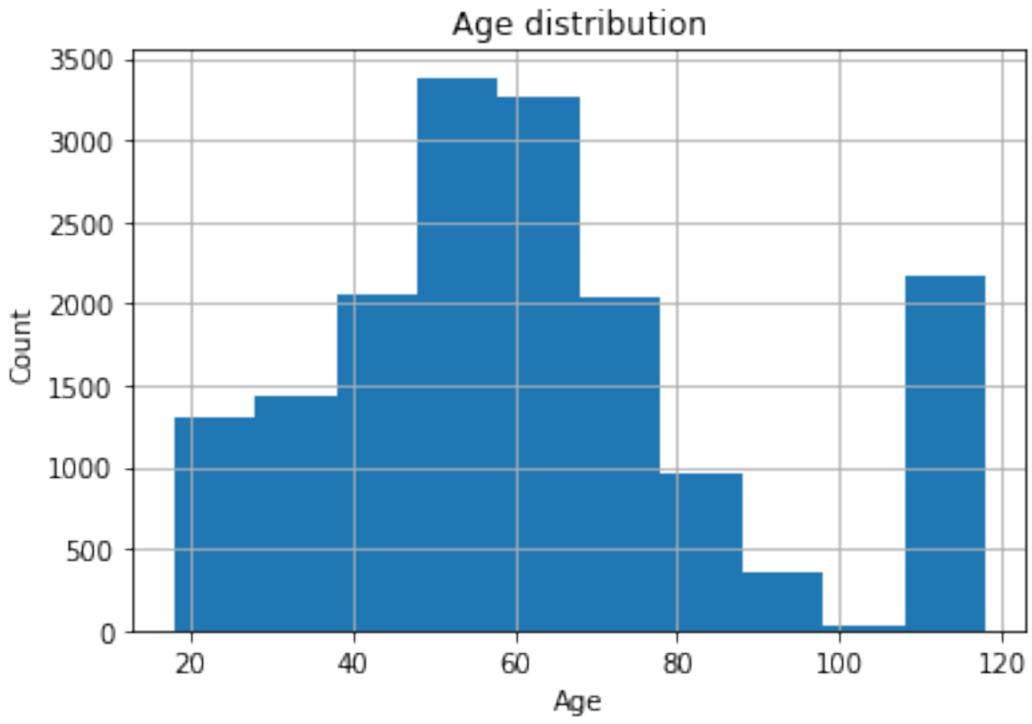
While looking at profile dataset, there are several features which can be analyzed with visualization. First is the income distribution among all customers. As shown below, the income has a nearly normal distribution with mean at around 65000.



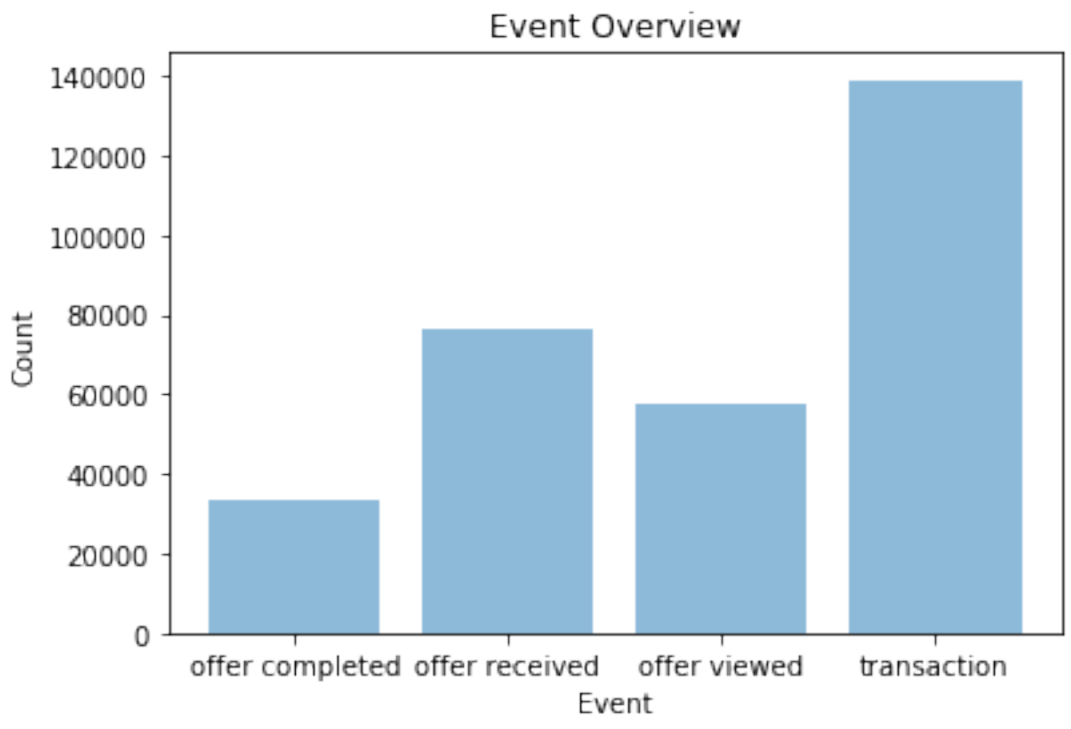
For gender, except missing value which will be removed later before building the model, the ‘O’ is a minority group, and the female and male group are quite close as shown below.



Another feature which is worth to take a look is age column, which turns out except the missing age encoded as 118 (which will be removed later) the other data distributes quite normal.



Next look at transcript dataset, as shown below, all event types within the dataset have even proportion except transaction which accounts a little more part of the dataset.



Algorithms and Techniques

To solve the problem stated previously, the project will first do some data preprocessing which aims to abstract the information within transcript dataset, then combine the transcript dataset to portfolio and customer information. A combined dataset will be helpful to the next steps. Then, feature engineering will be applied to get some more useful features to help building more accurate machine learning model such as time since the user becomes a member, number of offers received, number of transactions completed, etc.

After getting the dataset prepared, the next step is building the model. Here will try simple decision tree classifier and random forest. By comparing the performance, the better model will be selected based on which the further parameter tuning will implement.

Benchmark

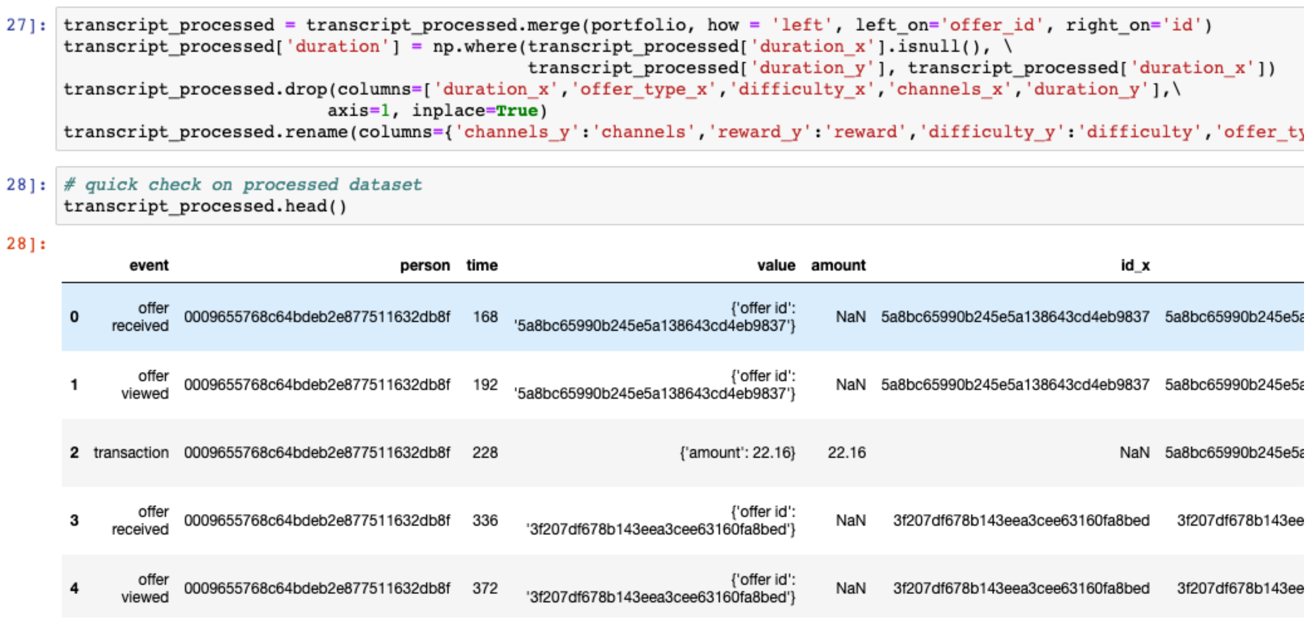
It's always a good practice to set a benchmark while building further fancy machine learning models. Here, in order to find out the best models, in this project, I'll initiate the simple decision tree classifier model as a baseline model. And I'll also use random forest as a comparison to baseline model to see if I could get a better accuracy.

III. Methodology

Data Preprocessing

In order to find out what mainly affect the finish of the transaction by sending the offer, in the data processing process, also need to process the data to merge the events of each specific offer sent so as to find out which offer was received, viewed and finally completed with a transaction.

Since offer_id is not associated with any 'transaction' event, in order to flag whether the offer has been finally completed with a transaction, here we need to link the offer id back to all transaction events. For BOGO and discount offer, both of them will have the consequence of offers received, viewed, transaction and offer completed which will apparently show that the offer is redeemed and should definitely be sent out. For the information offer, though there's no reward step there should still be a transaction that is linked to the usage of the offer.



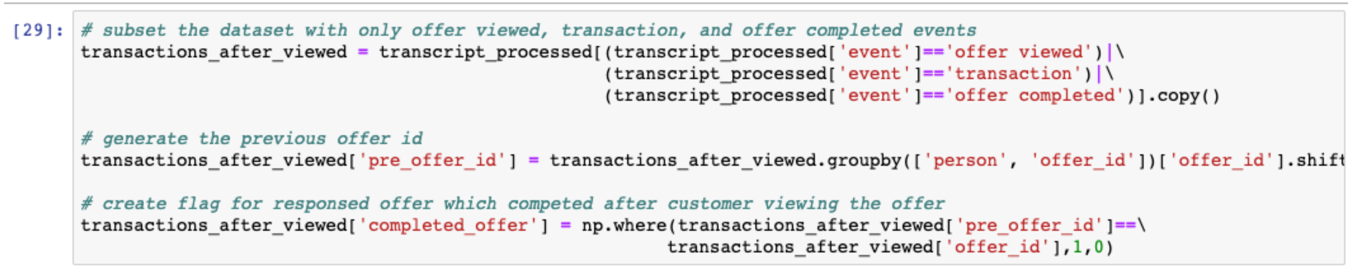
```
27]: transcript_processed = transcript_processed.merge(portfolio, how = 'left', left_on='offer_id', right_on='id')
     transcript_processed['duration'] = np.where(transcript_processed['duration_x'].isnull(), \
                                                 transcript_processed['duration_y'], transcript_processed['duration_x'])
     transcript_processed.drop(columns=['duration_x','offer_type_x','difficulty_x','channels_x','duration_y'],\
                              axis=1, inplace=True)
     transcript_processed.rename(columns={'channels_y':'channels','reward_y':'reward','difficulty_y':'difficulty','offer_ty
```

```
28]: # quick check on processed dataset
     transcript_processed.head()
```

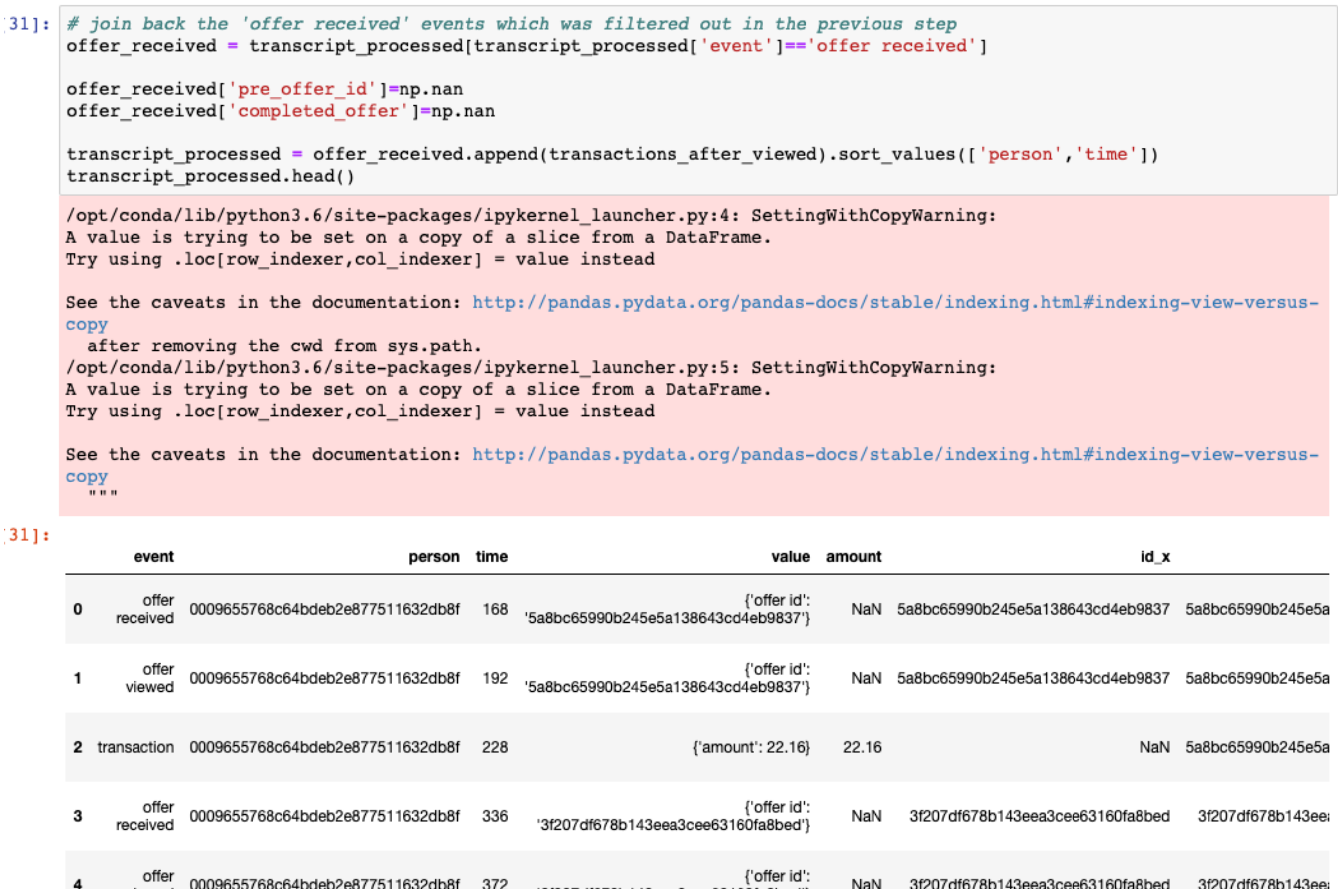
28]:

	event	person	time	value	amount	id_x	
0	offer received	0009655768c64bdeb2e877511632db8f	168	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	5a8bc65990b245e5a
1	offer viewed	0009655768c64bdeb2e877511632db8f	192	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	5a8bc65990b245e5a
2	transaction	0009655768c64bdeb2e877511632db8f	228	{'amount': 22.16}	22.16	NaN	5a8bc65990b245e5a
3	offer received	0009655768c64bdeb2e877511632db8f	336	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	NaN	3f207df678b143eea3cee63160fa8bed	3f207df678b143ee
4	offer viewed	0009655768c64bdeb2e877511632db8f	372	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	NaN	3f207df678b143eea3cee63160fa8bed	3f207df678b143ee

Next, after we get the data together, we need to extract the transactions which were completed after the offer was received and viewed. Since we've already filled all transaction's offer id, we can extract the transactions converted from offers by checking if the offer id before the transaction is the same as the transaction's offer id.



```
[29]: # subset the dataset with only offer viewed, transaction, and offer completed events
      transactions_after_viewed = transcript_processed[(transcript_processed['event']=='offer viewed')|\
                                                       (transcript_processed['event']=='transaction')|\
                                                       (transcript_processed['event']=='offer completed')].copy()

      # generate the previous offer id
      transactions_after_viewed['pre_offer_id'] = transactions_after_viewed.groupby(['person', 'offer_id'])['offer_id'].shift

      # create flag for responsed offer which competed after customer viewing the offer
      transactions_after_viewed['completed_offer'] = np.where(transactions_after_viewed['pre_offer_id']==\
                                                              transactions_after_viewed['offer_id'],1,0)
```



```
31]: # join back the 'offer received' events which was filtered out in the previous step
     offer_received = transcript_processed[transcript_processed['event']=='offer received']

     offer_received['pre_offer_id']=np.nan
     offer_received['completed_offer']=np.nan

     transcript_processed = offer_received.append(transactions_after_viewed).sort_values(['person','time'])
     transcript_processed.head()

     /opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
     A value is trying to be set on a copy of a slice from a DataFrame.
     Try using .loc[row_indexer,col_indexer] = value instead

     See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-
     copy
       after removing the cwd from sys.path.
     /opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
     A value is trying to be set on a copy of a slice from a DataFrame.
     Try using .loc[row_indexer,col_indexer] = value instead

     See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-
     copy
       """
```

31]:

	event	person	time	value	amount	id_x	
0	offer received	0009655768c64bdeb2e877511632db8f	168	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	5a8bc65990b245e5a
1	offer viewed	0009655768c64bdeb2e877511632db8f	192	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	5a8bc65990b245e5a
2	transaction	0009655768c64bdeb2e877511632db8f	228	{'amount': 22.16}	22.16	NaN	5a8bc65990b245e5a
3	offer received	0009655768c64bdeb2e877511632db8f	336	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	NaN	3f207df678b143eea3cee63160fa8bed	3f207df678b143ee
4	offer	0009655768c64bdeb2e877511632db8f	372	{'offer id':	NaN	3f207df678b143eea3cee63160fa8bed	3f207df678b143ee

Since the different offer has difference consequence of completion, for example, for the informational offer, there'll not be rewards. Therefore, here separate the transcript data by offer type (BOGO, discount, informational) for easier processing.

Within each offer type, use responded_offer flagged in previous steps we can filter out the offers which were successfully viewed and completed by users. For BOGO and discount offer, the responsed offer should be the one that with 'offer complete' events, and for the informational offer, just 'transaction' can be seen as a successful offer.

Next, will separate out customers who only viewed the offers without transaction and completion at the end and the customers who only received the offer without viewing it. Then, based on merged dataset, we can separate out customers who only viewed the offer after they received the offer and customers who didn't even open the offer after they receive the offer. Do the same steps for both BOGO and discount offer. After separating the different cases of customers, the following steps will firstly focus on customers who finish the transaction after receiving the offer and customers who only view the offer without any transaction. As for the informational offer, the offer could only be counted as responded under the effect of the offer when the transaction is finished within the duration of the offer.

Except basic data processing, basic feature engineering is also included here. There are several simple extended features which may help defining the model later. These features are the length of customer's membership, the count of the offer received for each user, the time lap between offers.

After steps above, merge the temporary data created above together, then drop the missing values in gender column, and split the channel column to the categorical variable using dummy variable.

Implementation

After pre-processing the data, the next step we'll start to implement models to figure out which factors affect most whether the customer will respond to the offer or not. And this project also attempts to predict whether the customer will respond to the different types of offers or not.

Therefore, we'll use the 'offer_responded' flag in the dataset to build models to predict if the customer will respond to the offer of not. Here we will choose the basic tree model as a baseline which will help explain the feature importance better so that we can get some insight into what factors affect customer's behavior most.

The implementation needs to first preprocess the dataset as what mentioned in preprocessing section (remove missing value, merge dataset, make categorical column dummy variables), then generate the features data frame and target vector. Then the data will be split into training and testing dataset which used for building model and predicting test.

After that, based on algorithm selected above, initiate the model object with default parameters used as baseline result. Apply the model to three types of offer using help functions and then do hyperparameter tuning to find out the best model for each type of offer.

Refinement

Since the project designing is to separate the dataset by different type of offers, it'll be much convenient to create function to make these steps into module.

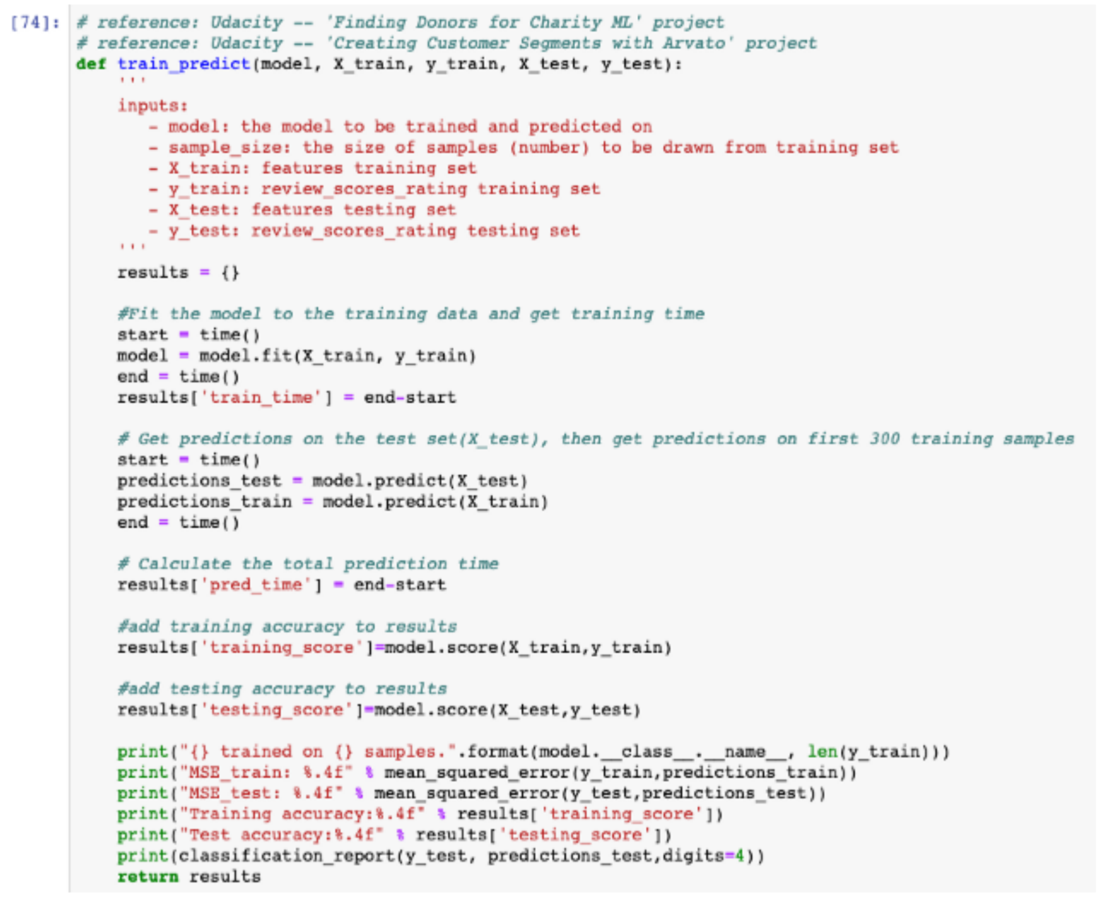
First create the function to prepare the dataset, process the features and target columns.

```
In [102]: def data_prep(df,drop_cols_prep):
              '''
              inputs:
              - df: prepared dataframe for modeling

              outputs:
              - Returns 2 dataframes - features and target dataframes
              '''
              # Split the data into features and target label
              target = df['offer_responded']
              features = df.drop(drop_cols_prep, axis=1, inplace=False)
              return features,target
```

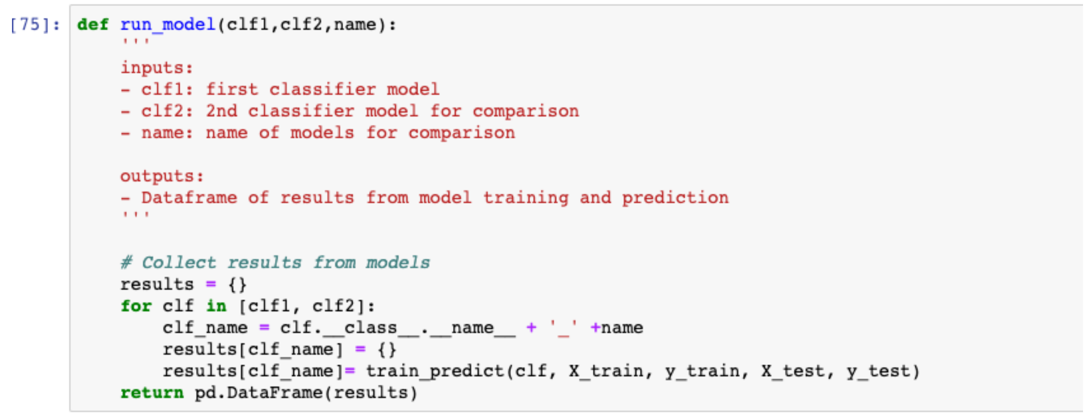
Since the implementation also needs split data into training and test sets, here create function to make this step reusable for each type of offer.

```
In [103]: def model_pipeline(features,target):
              '''
              inputs:
              - features & target dataframe

              outputs:
              - Splits features and target dataframe to train and test sets, performs feature scaling on both datasets.
              - Outputs X_train, X_test, y_train and y_test dataframes
              '''

              #split into training and test sets
              X_train, X_test, y_train, y_test = train_test_split(features,target, test_size=0.20, random_state=42)

              #fit and transform scaling on training data
              scaler=StandardScaler()
              X_train=scaler.fit_transform(X_train)

              #scale test data
              X_test=scaler.transform(X_test)
              return X_train,X_test,y_train, y_test
```

Also, for the step of model execution, since the same algorithm will be applied to each type of offer respectively, it’s a good practice to make the model execution into module.

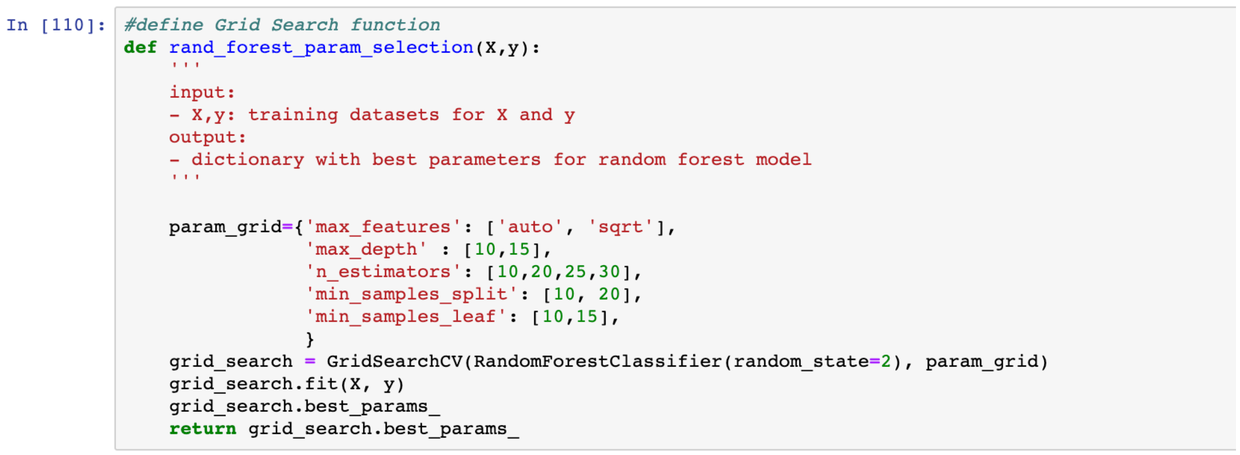


```
[74]: # reference: Udacity -- 'Finding Donors for Charity ML' project
      # reference: Udacity -- 'Creating Customer Segments with Arvato' project
      def train_predict(model, X_train, y_train, X_test, y_test):
          '''
          inputs:
             - model: the model to be trained and predicted on
             - sample_size: the size of samples (number) to be drawn from training set
             - X_train: features training set
             - y_train: review_scores_rating training set
             - X_test: features testing set
             - y_test: review_scores_rating testing set
          '''
          results = {}

          #Fit the model to the training data and get training time
          start = time()
          model = model.fit(X_train, y_train)
          end = time()
          results['train_time'] = end-start

          # Get predictions on the test set(X_test), then get predictions on first 300 training samples
          start = time()
          predictions_test = model.predict(X_test)
          predictions_train = model.predict(X_train)
          end = time()

          # Calculate the total prediction time
          results['pred_time'] = end-start

          #add training accuracy to results
          results['training_score']=model.score(X_train,y_train)

          #add testing accuracy to results
          results['testing_score']=model.score(X_test,y_test)

          print("{} trained on {} samples.".format(model.__class__.__name__, len(y_train)))
          print("MSE_train: %.4f" % mean_squared_error(y_train,predictions_train))
          print("MSE_test: %.4f" % mean_squared_error(y_test,predictions_test))
          print("Training accuracy:%.4f" % results['training_score'])
          print("Test accuracy:%.4f" % results['testing_score'])
          print(classification_report(y_test, predictions_test,digits=4))
          return results
```



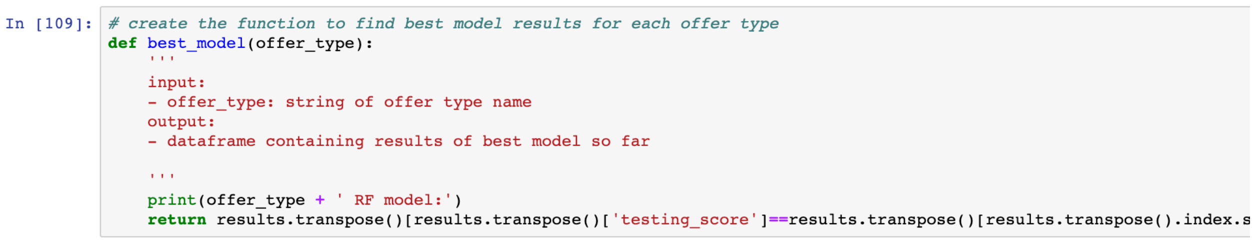
```
[75]: def run_model(clf1,clf2,name):
          '''
          inputs:
          - clf1: first classifier model
          - clf2: 2nd classifier model for comparison
          - name: name of models for comparison

          outputs:
          - Dataframe of results from model training and prediction
          '''

          # Collect results from models
          results = {}
          for clf in [clf1, clf2]:
              clf_name = clf.__class__.__name__ + '_' +name
              results[clf_name] = {}
              results[clf_name]= train_predict(clf, X_train, y_train, X_test, y_test)
          return pd.DataFrame(results)
```

Within the model tuning section, function to define Grid Search matrix is created to make the process more reusable.



```
In [110]: #define Grid Search function
          def rand_forest_param_selection(X,y):
              '''
              input:
              - X,y: training datasets for X and y
              output:
              - dictionary with best parameters for random forest model
              '''

              param_grid={'max_features': ['auto', 'sqrt'],
                          'max_depth' : [10,15],
                          'n_estimators': [10,20,25,30],
                          'min_samples_split': [10, 20],
                          'min_samples_leaf': [10,15],
                          }
              grid_search = GridSearchCV(RandomForestClassifier(random_state=2), param_grid)
              grid_search.fit(X, y)
              grid_search.best_params_
              return grid_search.best_params_
```

Also, function is created to find best model results for each offer type.

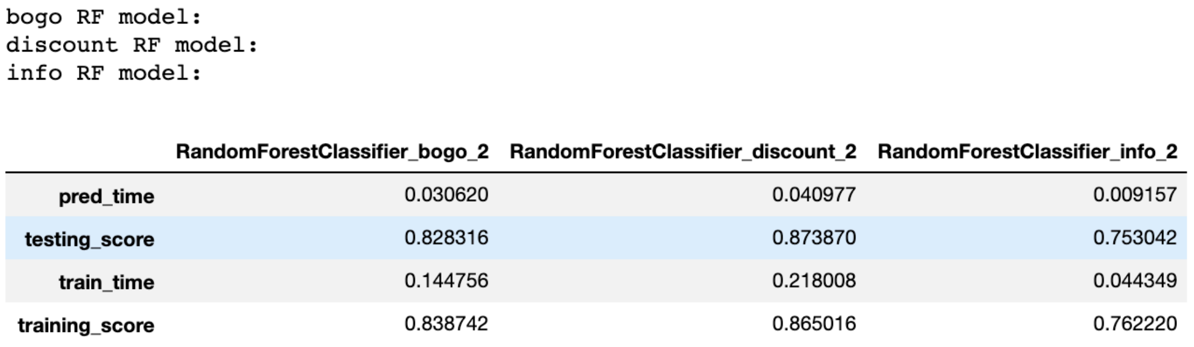


```
In [109]: # create the function to find best model results for each offer type
          def best_model(offer_type):
              '''
              input:
              - offer_type: string of offer type name
              output:
              - dataframe containing results of best model so far

              '''
              print(offer_type + ' RF model:')
              return results.transpose()[results.transpose()['testing_score']==results.transpose()[results.transpose().index.s
```

IV. Results

Model Evaluation and Validation

After getting the optimized parameters, rerun the model on each type of offer's dataset.



```
bogo RF model:
discount RF model:
info RF model:
```

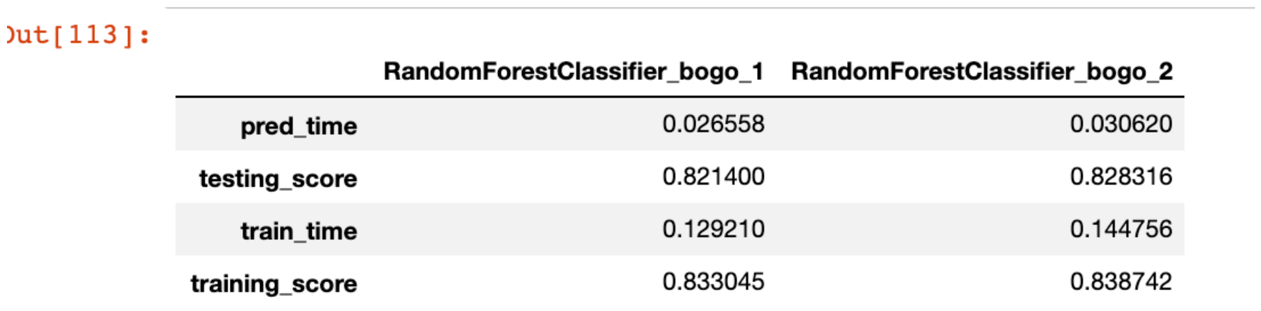
	RandomForestClassifier_bogo_2	RandomForestClassifier_discount_2	RandomForestClassifier_info_2
pred_time	0.030620	0.040977	0.009157
testing_score	0.828316	0.873870	0.753042
train_time	0.144756	0.218008	0.044349
training_score	0.838742	0.865016	0.762220

Justification

At first round, the default parameters were used before hyperparameter tuning to set as baseline comparison. By comparing the baseline model to the tuned one, the tune model gets better performance.

Note: *_1 represents the baseline and *_2 represents tuned result

The BOGO offer:

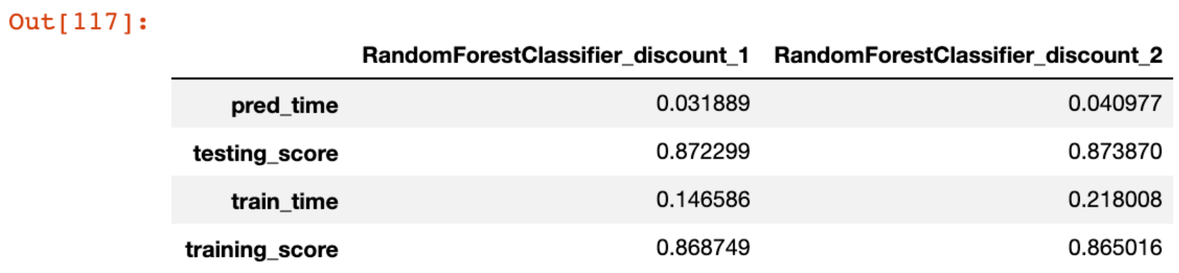


)ut[113]:

	RandomForestClassifier_bogo_1	RandomForestClassifier_bogo_2
pred_time	0.026558	0.030620
testing_score	0.821400	0.828316
train_time	0.129210	0.144756
training_score	0.833045	0.838742

As shown above in the comparison, after using tune parameters, the test accuracy slightly improved from 0.833 to 0.838 and the F1 score increase from 0.759 to 0.779.

The discount offer:

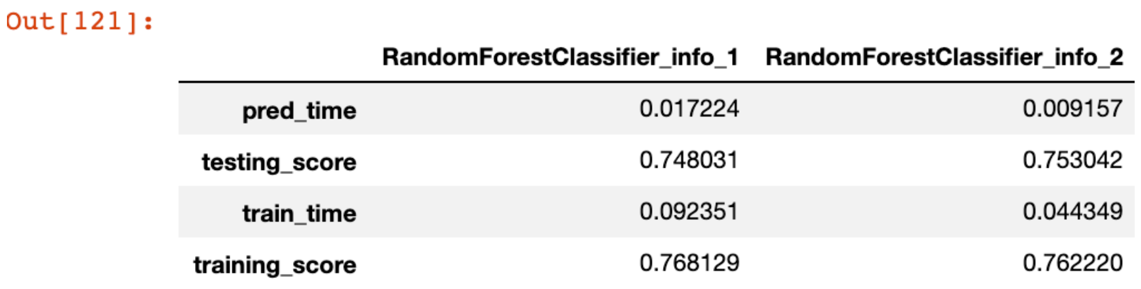


Out[117]:

	RandomForestClassifier_discount_1	RandomForestClassifier_discount_2
pred_time	0.031889	0.040977
testing_score	0.872299	0.873870
train_time	0.146586	0.218008
training_score	0.868749	0.865016

As shown above in the comparison, after using tune parameters, the test accuracy slightly improved from 0.872 to 0.873 and the F1 score increase from 0.814 to 0.816.

The informational offer:



Out[121]:

	RandomForestClassifier_info_1	RandomForestClassifier_info_2
pred_time	0.017224	0.009157
testing_score	0.748031	0.753042
train_time	0.092351	0.044349
training_score	0.768129	0.762220

As shown above in the comparison, after using tune parameters, the test accuracy slightly improved from 0.748 to 0.753 and the F1 score increase from 0.681 to 0.678.

V. Conclusion

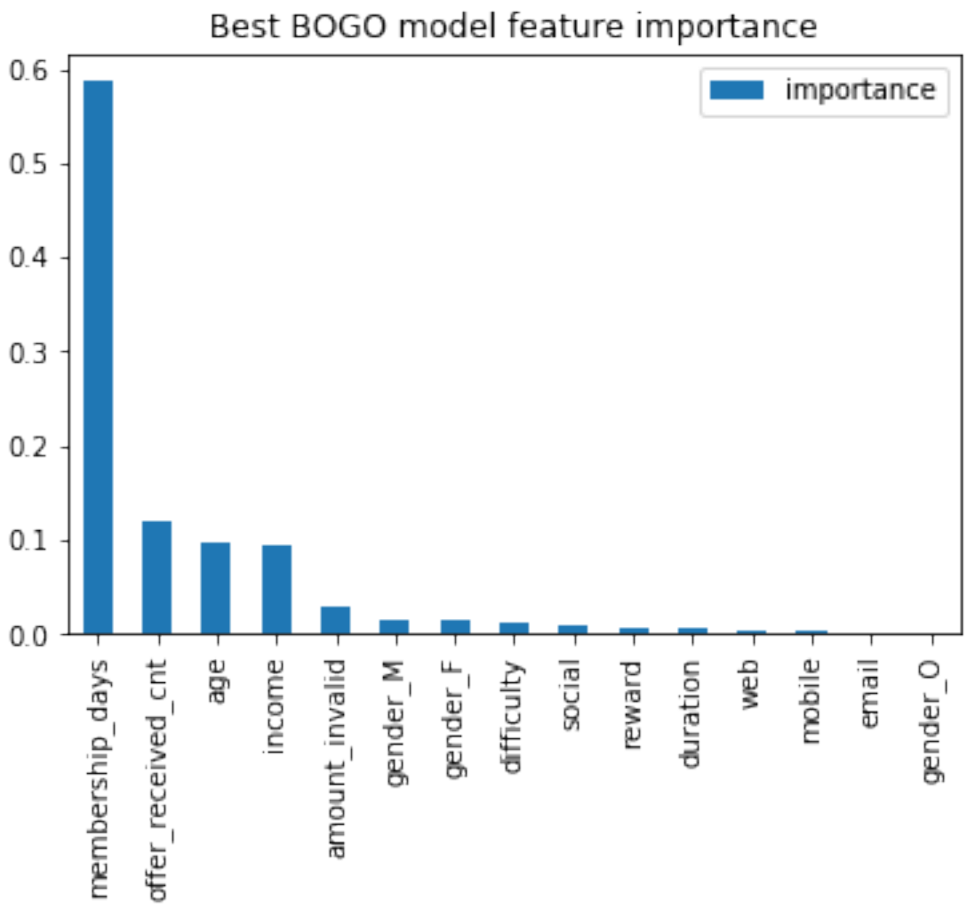
Reflection

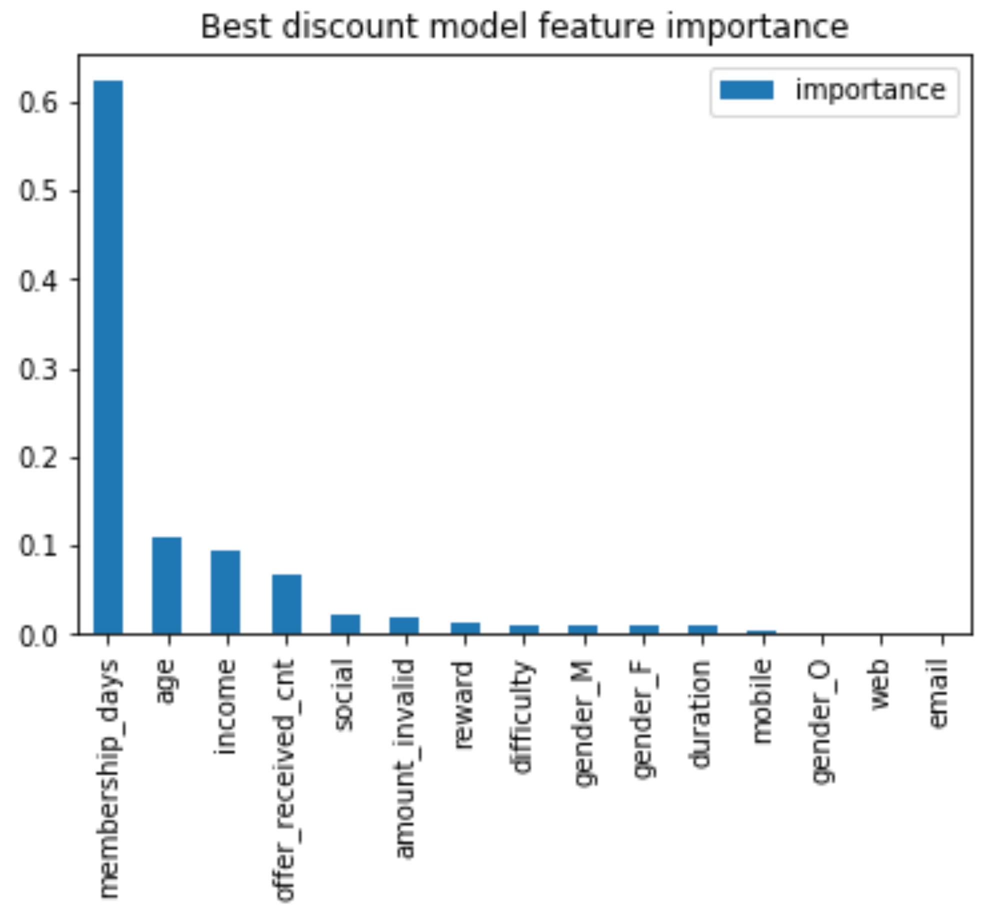
This project is trying to figure out:

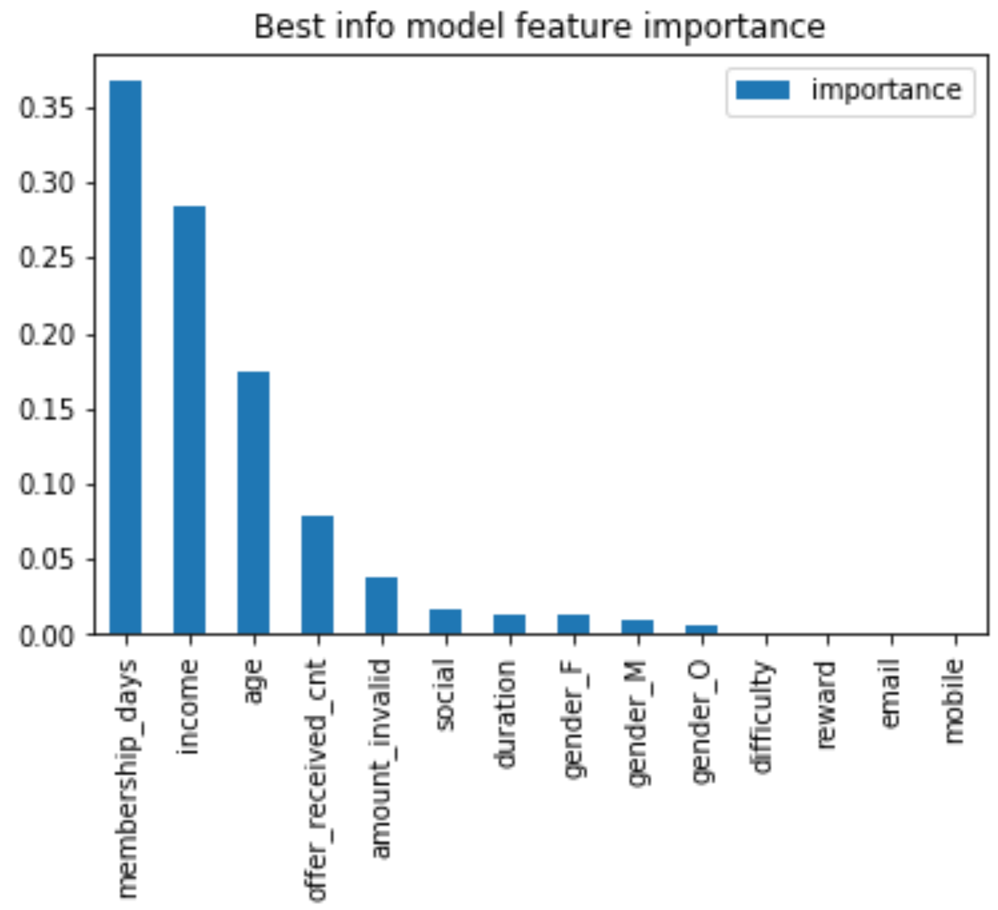
- What factors mainly affect the usage of the offer from the customer? Should the company send out the offer or not?
- How possible will a customer open and use the offer sent to them? Are there any common characteristics of the customers who take the offer?

By training and tuning the model on three type of offers' dataset, the final tuned models get quite good performance with BOGO 0.838, discount 0.873 and informational 0.753. With the machine learning model, we can know how possible the customer will open, view and finally complete the offer after the offer sent out.

But it's also important to know what factors mainly affect the usage of the offer so that in the future the company can utilize those insight to improve the possibility of offer completion. By looking at the feature importance of the tuned model as following:


Best BOGO model feature importance
importance
0.0
0.1
0.2
0.3
0.4
0.5
0.6
membership_days
offer_received_cnt
age
income
amount_invalid
gender_M
gender_F
difficulty
social
reward
duration
web
mobile
email
gender_O


Best discount model feature importance
importance
0.6
0.5
0.4
0.3
0.2
0.1
0.0
membership_days
age
income
offer_received_cnt
social
amount_invalid
reward
difficulty
gender_M
gender_F
duration
mobile
gender_O
web
email



As shown above, we can see that for all three types of offer, the most important factor that largely affects if the offer will be responded to eventually is the length of membership. That is, the longer the customer as a member of Starbucks, the more likely (s)he will respond to the offer they receive. Then the second and third important factors which affect the possibility of customer's response are age and income which very make sense. Also, the number of offers they received will also affect the response a lot.

Improvement

It's worthwhile to try some other enhancement in the step of model tuning. For example, probably, we can do some more experiment on feature engineering step to see if any other new features can improve the model, also I could also try to reduce some feature to see how it will affect the model performance.

Also, so far, the analysis is focused more on customer's who successfully finish the transaction after they received the offer, there should be more insight for the other cases where the customer finishes the transactions regardless of the offer. If we could get any insight into those cases, maybe we can send out more offers to those customers.

In addition, maybe we could do some unsupervised learning on clustering the customers based on information we are given, to see if there are any specific characteristics on a group of customers who will be more likely to respond to the offer.